Deep-Learning---Exp6

DL- Developing a Deep Learning Model for NER using LSTM

AIM

To develop an LSTM-based model for recognizing the named entities in the text.

THEORY

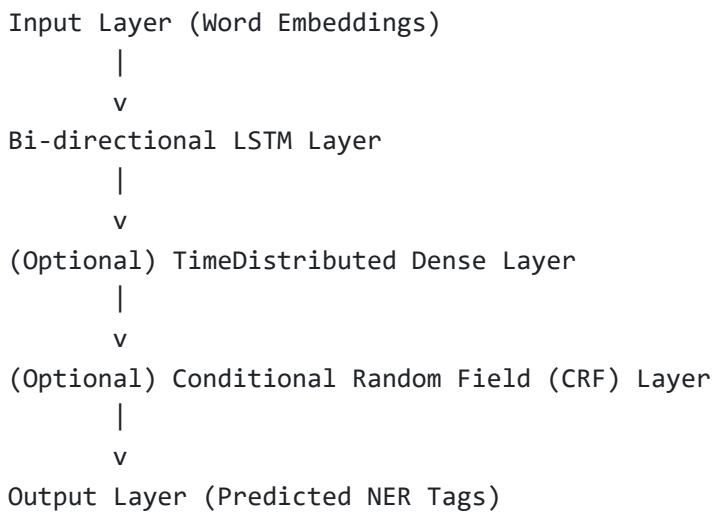
1. We aim to develop an LSTM-based neural network model using Bidirectional Recurrent Neural Networks for recognizing the named entities in the text.

2. The dataset used has a number of sentences, and each words have their tags.

3. We have to vectorize these words using Embedding techniques to train our model.

4. Bidirectional Recurrent Neural Networks connect two hidden layers of opposite directions to the same output.

Neural Network Model



DESIGN STEPS

STEP 1: Import the necessary packages.

STEP 2: Read the dataset and fill the null values using forward fill.

STEP 3: Create a list of words and tags. Also find the number of unique words and tags in the dataset.

STEP 4: Create a dictionary for the words and their Index values. Repeat the same for the tags as well.

STEP 5: We done this by padding the sequences and also to acheive the same length of input data.

STEP 6: We build the model using Input, Embedding, Bidirectional LSTM, Spatial Dropout, Time Distributed Dense Layers.

STEP 7: We compile the model to fit the train sets and validation sets.

PROGRAM

Name: Sharukesh T **Register Number:** 2305002022

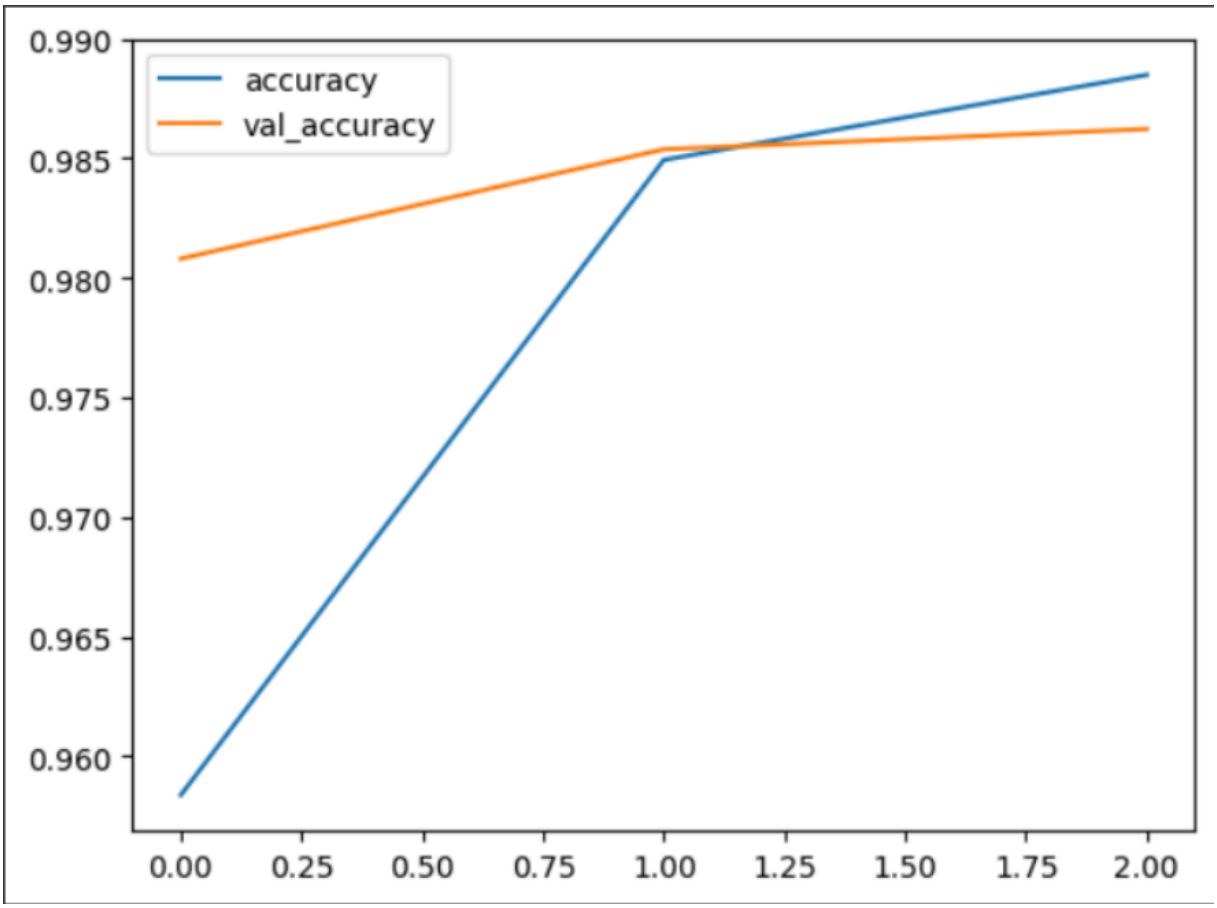
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from tensorflow.keras.preprocessing import sequence
from sklearn.model_selection import train_test_split
from keras import layers
from keras.models import Model

data = pd.read_csv("ner_dataset.csv", encoding="latin1")
data.head(50)
data = data.fillna(method="ffill")
data.head()

print("Unique words in corpus:", data['Word'].nunique())
print("Unique tags in corpus:", data['Tag'].nunique())
words=list(data['Word'].unique())
words.append("ENDPAD")
tags=list(data['Tag'].unique())

print("Unique tags are:", tags)
num_words = len(words)
num_tags = len(tags)
num_words

class SentenceGetter(object):
    def __init__(self, data):
        self.n_sent = 1
        self.data = data
        self.empty = False
        agg_func = lambda s: [(w, p, t) for w, p, t in zip(s["Word"].valu
                                                           s["POS"].value
                                                           s["Tag"].value
        self.grouped = self.data.groupby("Sentence #").apply(agg_func)
```

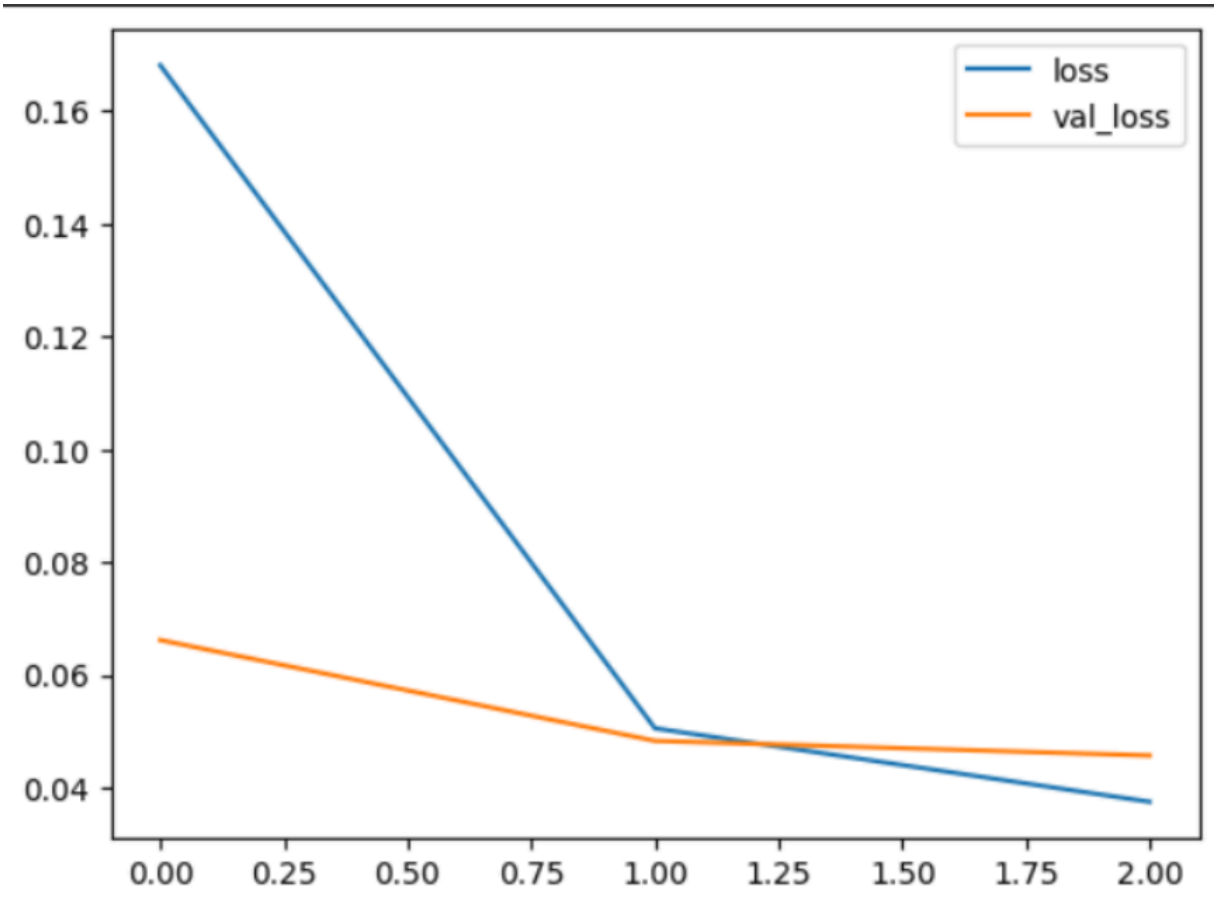
```
            self.sentences = [s for s in self.grouped]

        def get_next(self):
            try:
                s = self.grouped["Sentence: {}".format(self.n_sent)]
                self.n_sent += 1
                return s
            except:
                return None

    getter = SentenceGetter(data)
    sentences = getter.sentences

    len(sentences)
    sentences[0]
    word2idx = {w: i + 1 for i, w in enumerate(words)}
    tag2idx = {t: i for i, t in enumerate(tags)}
    word2idx

    plt.hist([len(s) for s in sentences], bins=50)
    plt.show()
    X1 = [[word2idx[w[0]] for w in s] for s in sentences]
    type(X1[0])
    X1[0]
    max_len=50

    X = sequence.pad_sequences(maxlen=max_len,
                      sequences=X1, padding="post",
                      value=num_words-1)
    X[0]
    y1 = [[tag2idx[w[2]] for w in s] for s in sentences]
    y = sequence.pad_sequences(maxlen=max_len,
                      sequences=y1,
                      padding="post",
                      value=tag2idx["O"])

    X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                        test_size=0.2, random
    X_train[0]
    y_train[0]

    input_word = layers.Input(shape=(max_len,))
    embedding_layer = layers.Embedding(input_dim = num_words,
                                       output_dim = 50,
                                       input_length = max_len)(input_word)
    dropout_layer = layers.SpatialDropout1D(0.13)(embedding_layer)
    bidirectional_lstm = layers.Bidirectional(layers.LSTM(
        units=250, return_sequences=True,recurrent_dropout=0.13))(dropout_lay
    output = layers.TimeDistributed(
        layers.Dense(num_tags, activation="softmax"))(bidirectional_lstm)

    model = Model(input_word, output)
    model.summary()
```

```
model.compile(optimizer="adam",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])

history = model.fit(
    x=X_train,
    y=y_train,
    validation_data=(X_test,y_test),
    batch_size=45,
    epochs=3,)

metrics = pd.DataFrame(model.history.history)
metrics.head()
metrics[['accuracy','val_accuracy']].plot()
metrics[['loss','val_loss']].plot()

i = 20
p = model.predict(np.array([X_test[i]]))
p = np.argmax(p, axis=-1)
y_true = y_test[i]
print("{:15}{:5}\t {}\n".format("Word", "True", "Pred"))
print("-" *30)
for w, true, pred in zip(X_test[i], y_true, p[0]):
    print("{:15}{}\t{}".format(words[w-1], tags[true], tags[pred]))
```

OUTPUT

Loss Vs Epoch Plot





Sample Text Prediction

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
Word            True      Pred

------------------------------
Officials       O         O
say             O         O
the             O         O
diplomats       O         O
were            O         O
to              O         O
discuss         O         O
the             O         O
recent          O         O
political       O         O
developments    O         O
in              O         O
Iraq            B-geo     B-geo
,               O         O
and             O         O
the             O         O
next            O         O
stages          O         O
of              O         O
the             O         O
political       O         O
process         O         O
including       O         O
the             O         O
drafting        O         O
of              O         O
the             O         O
nation          O         O
```

RESULT

Thus, an LSTM-based model for recognizing the named entities in the text is successfully developed.